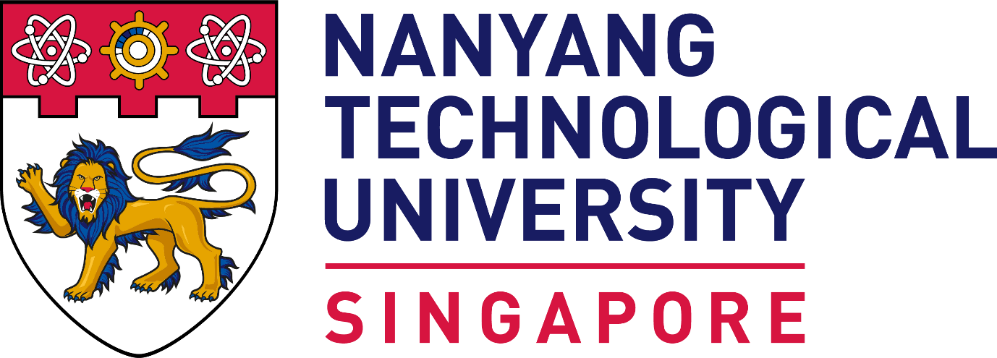
**music in air – digital theremin**



Submitted by

Wong Wee Meng (U1923221F)

**School of Computer Science and Engineering**

A Final Year Project presented to the Nanyang Technological University

in partial fulfilment of the requirements for the

Degree of Bachelor of Engineering

2021/22

# Contents

[Contents 1](#_Toc116839411)

[Table of Figure 3](#_Toc116839412)

[Abstract 4](#_Toc116839413)

[Acknowledgment 5](#_Toc116839414)

[1. Introduction 6](#_Toc116839415)

[1.1 Background 6](#_Toc116839416)

[1.1.1 Theremin 6](#_Toc116839417)

[1.1.2 Leap Motion 6](#_Toc116839418)

[1.2 Problem 7](#_Toc116839419)

[2. Existing Approach (Literature Review) 8](#_Toc116839420)

[3. Research Hypothesis 9](#_Toc116839421)

[3.1 Purpose 9](#_Toc116839422)

[The purpose of this project is to explore how well Unity Software can simulate playing the Theremin and to create a performance workbench exhibiting the best velocity achievable without any audio flaws. Then, we investigate if CSound can be utilized to enhance the sound quality and variety at an expense of latency. 9](#_Toc116839423)

[3.2 Scope 9](#_Toc116839424)

[4. Research and development plan 10](#_Toc116839425)

[4.1 Gemini 10](#_Toc116839426)

[4.2 Unity 10](#_Toc116839427)

[4.2.1 Unity’s Audio Function 11](#_Toc116839428)

[4.3 Csound Unity 12](#_Toc116839429)

[4.3.1 Csound Syntax 12](#_Toc116839430)

[4.3.2 Implement Csound with Unity 13](#_Toc116839431)

[5. Design and Implementation 14](#_Toc116839432)

[5.1 Using Mouse to generate sound 14](#_Toc116839433)

[5.1.1 Mouse Input 14](#_Toc116839434)

[5.1.2 Frequency Control 14](#_Toc116839435)

[5.1.3 Amplitude Control 15](#_Toc116839436)

[5.1.4 Generate Sound using OnAudioFilterRead 15](#_Toc116839437)

[5.2 Integrating the Leap Motion 16](#_Toc116839438)

[5.2.1 Leap Motion Input 16](#_Toc116839439)

[5.2.2 Mapping different frequencies based on hand position 16](#_Toc116839440)

[5.2.3 Mapping amplitude based on hand position 17](#_Toc116839441)

[5.3 Playing different wave functions 18](#_Toc116839442)

[5.4 User Interface 19](#_Toc116839443)

[5.5 Csound Implementation 20](#_Toc116839444)

[5.5.1. Basic Oscillation 20](#_Toc116839445)

[5.5.2 Sound Font Implementation 21](#_Toc116839446)

[5.5.3. FM (frequency modulated) synthesis with CSound 22](#_Toc116839447)

[5.5.4. WG (Wave Guided) synthesis with CSound 23](#_Toc116839448)

[5.5.5. Update User Interface 24](#_Toc116839449)

[5.6 Customised Starting Frequency 24](#_Toc116839450)

[6. Implementation Issues 25](#_Toc116839451)

[6.1 Lag Issues 25](#_Toc116839452)

[6.2 Hardware Issue 25](#_Toc116839453)

[7. Performance Measurement 26](#_Toc116839454)

[7.1 Procedures for measuring performance 26](#_Toc116839455)

[7.2 Results of performance between Unity default library and Csound library 26](#_Toc116839456)

[8. Conclusion 27](#_Toc116839457)

[9. References 28](#_Toc116839458)

# Table of Figure

[Figure 1 Theremin appearance 6](#_Toc116849965)

[Figure 2. Leap Motion Controller 6](#_Toc116849966)

[Figure 3. Ultraleap hand tracking interaction zones 7](#_Toc116849967)

[Figure 4. Software implemented by former students 8](#_Toc116849968)

[Figure 5. Physics representations of hands and VR controllers with interaction in Unity 10](#_Toc116849969)

[Figure 6. Generating a sine wave with the OnAudioFilterRead function 11](#_Toc116849970)

[Figure 7. An example of csd file 12](#_Toc116849971)

[Figure 8. Calculating size of each note in playing area 14](#_Toc116849972)

[Figure 9. Computing fixed and continuous frequency 14](#_Toc116849973)

[Figure 10. Computing amplitude based on mouse position 15](#_Toc116849974)

[Figure 11. Input frequency and amplitude value into OnAudioFilterRead() function to generate sound 15](#_Toc116849975)

[Figure 12. A software implementation of using mouse controller input to produce sound in Unity 15](#_Toc116849976)

[Figure 13. Computing the position of both hands in playing space 16](#_Toc116849977)

[Figure 14. Computing Frequency based on right hand position 16](#_Toc116849978)

[Figure 15. Computing Amplitude based on left hand position 17](#_Toc116849979)

[Figure 16. Smoothing the transition of amplitude to prevent pop sound in the buffer. 17](#_Toc116849980)

[Figure 17. Selecting between 6 different wave functions 18](#_Toc116849981)

[Figure 18. Interaction between main script and user interface 19](file:///D:\FYP\FYP%20Report.docx#_Toc116849982)

[Figure 19. Overall Software's User Interface 19](#_Toc116849983)

[Figure 20. Interaction between main game unity script and CsoundUnity script 20](file:///D:\FYP\FYP%20Report.docx#_Toc116849984)

[Figure 21. Using the 'Oscil' Opcode to generate sound 20](#_Toc116849985)

[Figure 22. Passing data between LibrarySound.cs and csd file 20](file:///D:\FYP\FYP%20Report.docx#_Toc116849986)

[Figure 23. Reading Sound Font into csd file 21](#_Toc116849987)

[Figure 24. A csd file that generate sound similar to a percussion flute 22](#_Toc116849988)

[Figure 25. A csd file that generate a sound similar to a clarinet 23](#_Toc116849989)

[Figure 26 Additional User Interface for user to toggle between Unity default library and CSound library. 24](#_Toc116849990)

# Abstract

Past FYP students had successfully implemented a digital theremin using the Leap Motion controller. Tools such as Microsoft Visual Studio, .NET Framework, NAudio library and Csound were used to implement the digital Theremin. However, static noises are also introduced as well. The reason of the occurrence of static noise is unknown, it could be a combination of NAudio and old drivers of Leap Motion.

Here, we recreate the same project using Unity to determine if the static sounds are removed. Then, we also introduce a performance workbench displaying the best velocity achievable without any audible flaws.

Investigating how CSound might produce a better sound is also discussed here. The performance benchmark is reviewed again in this area to ensure that the velocity do not decline too much in exchange for a better-quality sound.

Lastly, this paper discussed the user experience of the software based on user study to determine the impact of the software.

# Acknowledgment

I would like to express my gratitude and appreciation to my project supervisor, Associate Professor Alexei Sourin for his encouragement and guidance throughout the project.

# Introduction

## Background

This section discusses the background information of the Theremin, and the importance of simulating the Theremin with the Leap Motion.

### Theremin

The Theremin is a unique musical instrument that is played without physical contact. Moving one's hands near a vertical antenna causes the theremin to produce a different pitch, and a separate horizontal antenna controls the theremin's volume. However, precise control involves the feedback of hearing the theremin's tone and moving one's hands in accordance. Experienced players can roughly estimate the positions needed to produce various notes [1].



Figure Theremin appearance

### Leap Motion

The Leap Motion Controller is an optical hand tracking module that captures hand movements with exceptional accuracy. The Leap Motion Controller features an interaction zone that spans from the device in a 140x120° typical field of vision and extends from 10cm to 60cm or more [2] .

A picture containing text, person, indoor, computer

Description automatically generated

Figure . Leap Motion Controller

A picture containing diagram

Description automatically generated

Figure . Ultraleap hand tracking interaction zones

## Problem

The three main negative factors affecting usage of Theremins are affordability, portability, and user experience.

Theremins typically cost between $200 and $899 [3], which is quite pricey when compared to the Leap Motion's pricing of about $80 [4]. Furthermore, the Theremin is larger in shape compared to a small Leap motion controller. It becomes less portable because of this.

Lastly, playing the Theremin may results a poor user experience because it is difficult to play, and most users would give up after a while. The user would not be able to distinguish the note that is sounded without having perfect pitch, which is rare ability to recognize specific musical note without the need of a reference tone. However, with the use of Leap Motion, a simple software can be developed that displays the note being played by the user. This may improve user experience as the user knows the note which he is currently playing.

Therefore, it is important to simulate the Theremin digitally with a Leap motion as it would benefit the users by providing an affordable and portable option, as well as improving the user experience.

# Existing Approach (Literature Review)

Theremin simulation software had been implemented by former students using .NET Framework using C# programming language. Depending on the user's hand location in relation to the Leap Motion controller, the software can generate different frequencies using the NAudio sound libraries. For various audio effects, users can also choose from several wave functions like sine, triangle, and sawtooth. Additionally, there is a simple user interface that shows an image of a piano keyboard for reference and the note that is currently being played.

A screenshot of a computer

Description automatically generated with medium confidence

Figure . Software implemented by former students

However, there are some problems with the software. Static noise and popping sounds are produced on around every 2 seconds while playing the software which results to a bad user experience. The origin of the static noise is not known, and it could be due to the NAudio sound library used or the old drivers of the leap motion.

Thus, there is a need to explore an alternative way to simulate playing the Theremin and create high quality sound.

# Research Hypothesis

## Purpose

## The purpose of this project is to explore how well Unity Software can simulate playing the Theremin and to create a performance workbench exhibiting the best velocity achievable without any audio flaws. Then, we investigate if CSound can be utilized to enhance the sound quality and variety at an expense of latency.

## 3.2 Scope

In this project, instead of developing a program that generates sound from scratch, we will utilise existing sound library to ease the generation of sound. Should the popping sound persist even when using a new sound library, we will proceed to explore other hardware alternative like Kinect as we would not investigate the hardware issue of the Leap Motion Controller.

# Research and development plan

This section discussed the necessary research and knowledge required for the development of the software.

## 4.1 Gemini

There is a recent released of [Gemini](https://developer.leapmotion.com/tracking-software-download), the fifth version of UltraLeap hand tracking software. With a newer version, there is an improved in two-handed interaction, initialization and allows more robust and accurate hand tracking. This would also allow a more stable and reliable hardware to work with.

## 4.2 Unity

In this project, the Unity software is being explored. Unity is a cross-platform game engine that support a variety of desktop, mobile, console and virtual reality platforms. The ultra-leap plugin for unity provides the tools and utilities to connect Unity applications to hand tracking data makes it possible to use with the leap motion.

Graphical user interface

Description automatically generated with medium confidence

Figure . Physics representations of hands and VR controllers with interaction in Unity

Before implementing leap motion in unity, the ultra-leap hand tracking software must be installed first, followed by the unity modules package which includes Core, Interaction Engine, and the Hands Module.

### 4.2.1 Unity’s Audio Function

Unity has a main default audio function which is known as OnAudioFilterRead. Every time a chunk of audio is supplied to the filter, the OnAudioFilterRead function is invoked. The audio data is a float array ranging from -1 to 1 that contains audio from the preceding filter in the chain or the Audio Clip on the Audio Source [5]. The frequency that we had computed in every frame is used in the OnAudioFilterRead function to generate a sine wave.

Text

Description automatically generated

Figure . Generating a sine wave with the OnAudioFilterRead function

## 4.3 Csound Unity

Csound is a programming language (based on C) that was created and optimized for sound rendering and signal processing. Currently, there is a custom Csound package for Unity known as CsoundUnity. It provides Unity users to access the Csound's core API within their Unity C# scripts and provides a Unity bridge to a set of pInvoke signatures that are used to call Csound's native API functions from C# code. Once the CsoundUnity packages is downloaded, it needs to be imported directly into Unity.

For more documentation of Csound can be found here. http://rorywalsh.github.io/CsoundUnity/html/class\_csound\_unity.html#af950ce82e184ee08a01a59e007f25f12

### 4.3.1 Csound Syntax

Csound consists of 2 different files, an orchestra file and a score file. The figure below is an example of a Csound .csd script.

Graphical user interface, text

Description automatically generated

**Variables**

**Opcodes**

**Orchestra**

**Score**

Figure . An example of csd file

The Orchestra is defined above the score. The orchestra section starts with a header and defined the list of instruments below, and the score section contains tables events and instrument events. Anything appear at the right side of a semicolons are comments.

There are three types of variables, and they usually start with letter “a”, “k” and “i”. The letter ‘a’ means audio rate and “k” means control rate. They can be modified during runtime, except for variables that start with “i” which are initialized to a value at the start and do no change.

There are many opcodes available in Csound. Every opcode may have variables on the left or on the right of the opcode. In the figure above, the opcode “oscil” takes in 3 variables, ‘iamp’, ‘ifqc’ and ‘itabl1’, and output the variable ‘aout’. For more information on the available opcodes can be found here: <https://github.com/falkTX/cabbage/blob/master/Builds/MacOSX/opcodes.txt>.

## 4.3.2 Implement Csound with Unity

To implement Csound with Unity, two important scripts are commonly used.

First is CsoundUnity.cs which is the main Csound script. It creates an instance of CsoundUnityBridge for accessing most Csound's methods. CsoundUnity focus on common methods such as compiling and channel communication and does not expose all of Csound's native API functions. This script is attached to a GameObject and should be instantiated when a game first starts.

Second is the csd file which had been discussed above. This file provides some simple instrument definition and contains the orchestral and score information. Each CsoundUnity.cs will take in one csd file and the variables inside the csd file can be accessed mainly using SendScoreEvent() or the setChannel() method.

# Design and Implementation

## Using Mouse to generate sound

To begin with, we implemented a sound generation feature using a simple mouse input. A simple scene is set up with minimum interface.

### 5.1.1 Mouse Input

The position of the mouse will determine the amplitude and frequencies being played. Moving in the horizontal direction will controls the amplitude while moving in the vertical direction will controls the frequency of the sound. Input.mousePosition is called to retrieve the position of the Mouse.

### 5.1.2 Frequency Control

The next step is to map the mouse position to play different amplitude and frequency of sound. Firstly, we compute the total notes by multiplying total octave by 12. Example, if there are 3 octaves, then total notes would be 36. The size of each note determines the location of the note in the 3D playing area. It can be computed by dividing the height of the screen by the total notes.

Graphical user interface, text

Description automatically generated

Figure . Calculating size of each note in playing area

Next, we can compute the frequency by obtaining the note index. The note index determine what note is being played. For a 3-octave range, the note index will be between 0 to 36, example from A3 to G6 We then compute the power raised by dividing the note Index by 12, to get a value between 0 to 3. Assume the starting frequency is 220, a note Index of 0, 1, 2, 3 would result in a frequency of 220, 440, 880 and 1760 respectively. For a non-continuously frequency function, the noteIndex will be round down to an integer.

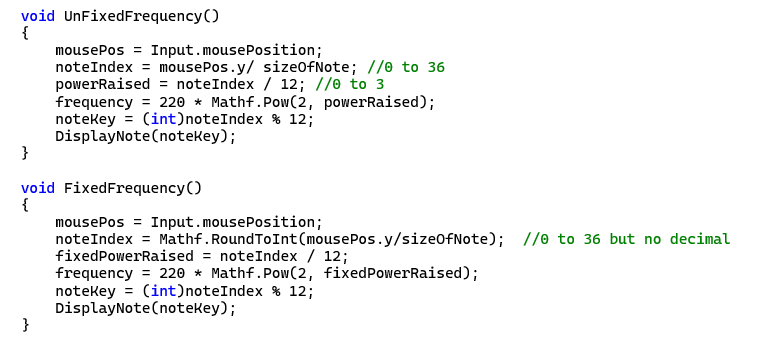


Figure . Computing fixed and continuous frequency

### 5.1.3 Amplitude Control

The amplitude of the sound is determined by the x position of the mouse in respective to the screen width.

Text, letter

Description automatically generated

Figure . Computing amplitude based on mouse position

### 5.1.4 Generate Sound using OnAudioFilterRead

With the computed amplitude and frequency, we can pass the values to the OnAudioFilterRead () function to generate the sine wave.

Text

Description automatically generated

Figure . Input frequency and amplitude value into OnAudioFilterRead() function to generate sound

Since this feature created a smooth generation of sound, the next step is to substitute the mouse input with a Leap Motion Controller Input.

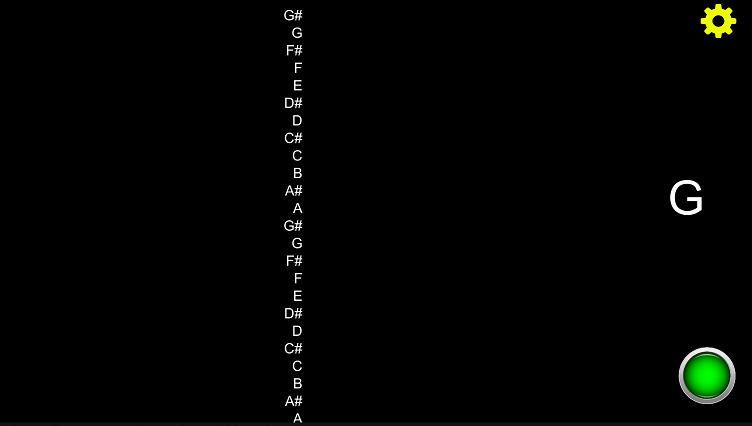


Figure . A software implementation of using mouse controller input to produce sound in Unity

## Integrating the Leap Motion

### 5.2.1 Leap Motion Input

Accessing the Leap Motion library is possible after importing the Leap package into Unity. Once imported, the 3D hand model can be view in the Game View. Simulating playing the Theremin with 2 hands would means that we must compute the position of both hands. We must first deduce the position of the hands in the 3D leap motion space. Since the right hand controls the pitch, the average y axis position of all the fingers is computed. The change of the y position in the 3D leap motion space will determine the frequency being produced. The thumb is excluded since the mobility of the thumb is limited and would not greatly affect the average position.

Similarly for the left hand, the average y axis position of all the fingers is computed, and it will determine the amplitude of the sound.

Graphical user interface, text, application, email

Description automatically generated

Figure . Computing the position of both hands in playing space

### 5.2.2 Mapping different frequencies based on hand position

The boundary of the playing space is set by initialising the minimum and maximum y position of the leap motion space. We then map the available range of the 3D space to the range of frequency. For example, we set the minimum y position to be 185 and maximum y position to be 293. If 3 octaves are selected, then the hand position at 185 would be note A2 and the hand position at 293 would be note G#5. Any position lies in the range will be from A2-G#7. The new frequency computation script will be now this.

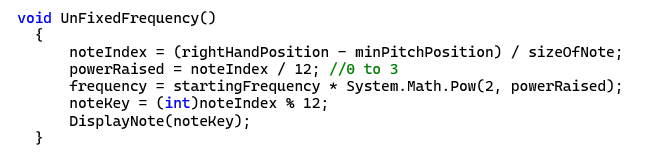


Figure . Computing Frequency based on right hand position

### 5.2.3 Mapping amplitude based on hand position

Instead of using mousePosition.x, we use the left-hand position in space to calculate the amplitude.

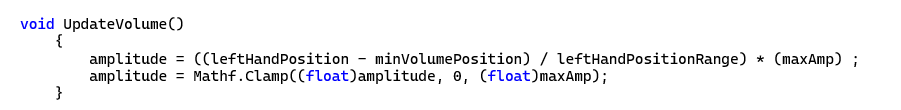


Figure . Computing Amplitude based on left hand position

There is a slight problem when generating sound. When the user removes their left hand in the play area, the sudden change of the amplitude value to zero will cause a pop sound. To tackle this issue, a interpolation is added for a smooth transition of the amplitude value to zero.

Graphical user interface, text, application

Description automatically generated

Figure . Smoothing the transition of amplitude to prevent pop sound in the buffer.

## Playing different wave functions

Six different wave formulas are implemented in this software. User can select different wave options from the dropdown menu in the game scene to be played by the OnAudioFilterRead() function.

Graphical user interface, text

Description automatically generated

Figure . Selecting between 6 different wave functions

## User Interface

Up till now, all the features are inside the UnitySoundLeap.cs script which manage the leap motion input, computing frequencies and amplitude, and output the sound. Therefore, to avoid increasing the complexity of this script, another script called UIHand.cs is created. The aim of this script is to manage the user input and update the screen accordingly and passing new data to the UnitySoundLeap.cs.

Figure 18. Interaction between main script and user interface

UIHand.cs

UnitySoundLeap.cs

User Interface

The user interface or this software is shown below. Users are able to:

1. Select a range of octave from 1 to 3.
2. Toggle between fixed and continuous frequency.
3. Toggle the display of keyboard guide and alphabet notes guide.
4. Select from a variety of wave function.
5. Close and open setting menu.

Graphical user interface

Description automatically generated with medium confidence

Figure . Overall Software's User Interface

## Csound Implementation

The Csound library is explored to have access to a much more variety of instrumental sound. Instead of using the OnAudioFilterRead () function to create sound, the csd file is responsible for creating sound. A LibrarySound.cs script is created to manage everything that is related to using Csound library.

The main script UnitySoundLeap.cs will pass the real time frequency and amplitude data to the LibrarySound.cs for updating the csd file from CSoundUnity.cs. The data flow is shown below.

LibrarySound.cs

UnitySoundLeap.cs

CSoundUnity.cs

Figure 20. Interaction between main game unity script and CsoundUnity script

### 5.5.1. Basic Oscillation

An basic instrument that output a sine wave function is defined in the .csd file shown in figure below.

Text

Description automatically generated

Figure . Using the 'Oscil' Opcode to generate sound

The variables kfreq and kamp controls the frequency and amplitude of the sound. Since the values will change during runtime, the LibrarySound.cs must be able to access the two variables in the csd file. This can be done using the SetChannel() method in Unity script and chnget in csd file. Below is an example on how to pass data from LibrarySound.cs to the csd file.

csd file

kfreq chnget “Frequency”

kamp chnget “Amplitude”

LibarySound.cs file

csoundUnity.SetChannel("Frequency", frequency);

csoundUnity.SetChannel("Amplitude", currentAmp);

Figure 22. Passing data between LibrarySound.cs and csd file

### 5.5.2 Sound Font Implementation

To have access to different instruments, the sf\_GMbank.sf2 is being explored. Sf\_GMbank.sf2 is a sound font that contains 329 presets instrument. There is a sample scene in the CsoundUnity package that illustrate playing the French horn instrument using the 60th preset index. The ‘inum’ variable in the csd file determines the midi number. As shown below, the variable ‘inum’ is initialised to 69, therefore playing the note “A” with 440Hz.

Text, letter

Description automatically generated

Figure . Reading Sound Font into csd file

The sound produced is beautiful however there is a limitation. In this project, the frequency of sound produced must be continuous. This csd file is unable to play all the frequency between inum 69 to 70, (A to A#). Thus, other approach is explored.

Instead of changing the inum, we can change the frequency instead. However, the variable frequency in this csd file is slightly different. It does not take in the hertz value. Instead, it is set as preset of 1. Therefore a inum of 69 and frequency of 1 means 440Hz, and a frequency of 2 would means 880Hz.

With this new approach, we can play continuous frequency, however the produced sound remains strange. Additionally, the author of this csd sample had commented in the script that we should not change the frequency parameter, and it should remain as 1. The only parmeter that can be changed is the inum variable which determine the midi number. Therefore, sound font is mainly for playing fixed frequency. It is more suitable for games that simulate playing a piano with each note already preset to different inum value.

### 5.5.3. FM (frequency modulated) synthesis with CSound

Since using sound font is not possible, the FM opcode is being explored. In frequency modulated synthesis, the instrument plays on a continuous frequency and is being changed overtime. One example of such opcode is fmpercfl, which uses FM synthesis to create a percussive flute sound. The syntax for this opcode is

ares fmpercfl kamp, kfreq, kc1, kc2, kvdepth, kvrate[, ifn1, ifn2, \ ifn3, ifn4, ivfn]

The kamp and kfreq represent the amplitude and frequency which we can be accessed from LibrarySound.cs. The remaining parameters can be adjusted to affect instrument sound such as vibrato depth and vibrator rate.

We had implemented a few instruments such as Percussion flute, Tublar bell and Hammond organ. The outcome is successful, and the sound produced sounded nicer than a normal sine wave. The percussion flute csd file is shown below.

Text

Description automatically generated

Figure . A csd file that generate sound similar to a percussion flute

### 5.5.4. WG (Wave Guided) synthesis with CSound

Apart from using FM synthesis, there are other methods explored to produce sound using wave guided synthesis. The wgclar opcode is explored here which allows us to create a tone like a clarinet, using a physical model developed from Perry Cook. The implemented csd file is shown below.

Text

Description automatically generated

Figure . A csd file that generate a sound similar to a clarinet

The kfreq and kamp variables will be passed from the LibrarySound.cs to control frequency and amplitude and the remaining parameter will determine how would the clarinet sound.

### 5.5.5. Update User Interface

With the implementation of Csound Library, more interfaces had been added. A new drop-down option to select between the different Csound instruments is implemented. Additionally, the option to toggle between Unity default library and Csound Library is also implemented.

Graphical user interface, text, application

Description automatically generated

Figure Additional User Interface for user to toggle between Unity default library and CSound library.

## 

## 5.6 Customised Starting Frequency

Currently, all instruments start from a frequency of 440Hz. This caused some instrument to sound strange. For example, a Tubular bell would sound strange if the frequency is below 880Hz.

Therefore, additional code is added into the LibrarySound.cs to be able to change the starting frequency variable in UnitySoundLeap.cs, depending on the instrument that the user had selected.

# Implementation Issues

## Lag Issues

Certain issues such as lags occurs when using the software, and the frame rate dropped to 10-20 frame per seconds (FPS). This usually happens when the user computer did not shutdown or restart for a few days. This is quite common even when developing games in Unity, and the fps is extremely low when the computer had operated for a long time. This issue can be solved by restarting the user computer and start open the software again. This allows the software to be able to run at a normal speed of 200-300 fps again.

## Hardware Issue

Occasionally, the leap motion will stop working when the software is developing and testing with Unity. The leap motion controller is not able to track anything and lose its connection. Plugging out and plugging in the leap motion controller would not solve the problem. The only solution is to restart Unity again and the leap motion controller can then be connected once again.

# Performance Measurement

Csound library proved to be successfully to be able to generate a more variety of sound in Unity. However, using an external library would introduce additional latency, therefore we aimed to investigate how much additional latency is introduced.

## Procedures for measuring performance

An experiment is conducted to assess the performance with BPM(beat per minutes) as the primary determinant. The song, ‘Twinkle Twinkle Little Star’ is composed using Garage Band with an iPad. The soundtrack can be looped in Garage Band at the desired tempo. The tempo is initially set to 150. We used the Unity default OnAudioFilterRead() function to play along with the soundtrack from the garage band. Each time we can play along with the garage band soundtrack, the tempo will increased by ten.

This continues until our software's latency caused it to be unable to keep up with the pace. The highest tempo will be recorded, and the experiment will then be repeated using the CSound Library.

## Results of performance between Unity default library and Csound library

# Conclusion

In conclusion, it is proven that we can replicate past project with Unity successfully without producing any audio flaws such as static noise and popping sound. We also had successfully integrating Csound library and introduced more variety of sound to be played. Even though implementing Csound had cause additional delay, it is non-negligible enough to be heard. The team will continue to tune the different parameters in the existing .csd file, as well as exploring more opcodes in Csound to generate more variety of sound.

# References

[1] K. Mathew, "Strange Vibrations: The Evolution of the Theremin," 2019.

[2] UltraLeap. "How Hand Tracking Works." <https://www.ultraleap.com/company/news/blog/how-hand-tracking-works/> (accessed 22 August 2022, 2022).

[3] T. World. "Theremins and Theremin Kits." <http://www.thereminworld.com/theremin-store> (accessed.

[4] D. Heaney. "Ultraleap Gemini Impressions: Great Hand Tracking Is Coming Soon." <https://uploadvr.com/ces-ultraleap-gemini-hand-tracking/#:~:text=By%202019%20even%20AltSpace%20dropped,egg%20problem%20of%20input%20accessories>. (accessed 21 August 2022, 2022).

[5] U. Technologies. "Unity Documentation." <https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnAudioFilterRead.html> (Accessed 16-10-2022).